Hanoi University of Science and Technology
School of Information and Communication Technology

PROJECT REPORT

Project 1 - IT3910E

Topic: **Image classification using Convolutional Neural Networks**

Instructor: PhD. Nguyen Nhat Quang

Student: Chu Hoang Duong - 20194429

Hanoi, January 2022

Abstract

Image classification problem is one of the most popular problems in Computer Vision. The main purpose of this project is to apply the concept of a Deep Learning algorithm namely, Convolutional neural networks (CNN) in image classification. I built a model from scratch and compared my proposed CNN model to the other models using different backbones (VGG16, ResNet 34, AlexNet, SqueezeNet, MobileNet V3).
The dataset used for training and testing is extended animal 10 [1] with 60.907 images divided into 20 classes.
The performance of the models is evaluated based on the quality metric known as Cross Entropy Loss, validation accuracy and testing accuracy. The experimental result analysis based on the quality metrics and the graphical representation proves that the models give fairly good classification accuracy for the tested dataset.

Table of content

I. Introduction

Convolutional Neural Network (CNN) which was first introduced in the 1980s is a class of artificial neural network, most commonly applied to analyze visual imagery. CNNs have application in *image and video recognition, recommender systems, image classification, image segmentation, medical image analysis, natural language processing, brain-computer interfaces and financial time series*.

CNNs are composed of multiple layers of artificial neurons. Artificial neurons, a rough imitation of their biological counterparts, are mathematical functions that calculate the weighted sum of multiple inputs and output an activation value.

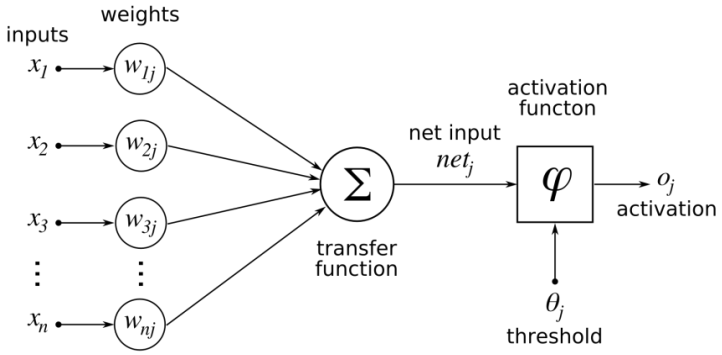


Figure 1.1: The structure of an artificial neuron, the basic component of artificial neural networks (source: Wikipedia)

The behavior of each neuron is defined by its weights. When fed with the pixel values, the artificial neurons of a CNN pick out various visual features.

When inputting an image into a CNN, each of its layers generates several activation maps. Activation maps highlight the relevant features of the image. Each of the neurons takes a patch of pixels as input, multiplies their color values by its weights, sums them up, and runs them through the activation function.

The operation of multiplying pixel values by weights and summing them is called "convolution" (hence the name convolutional neural network). A CNN is usually composed of several convolution layers, but it also contains other components. The final layer of a CNN is a classification layer, which takes the output of the final convolution layer as input.

Based on the activation map of the final convolution layer, the classification layer outputs a set of confidence scores (values between 0 and 1) that specify how likely the image is to belong to a class.[2]

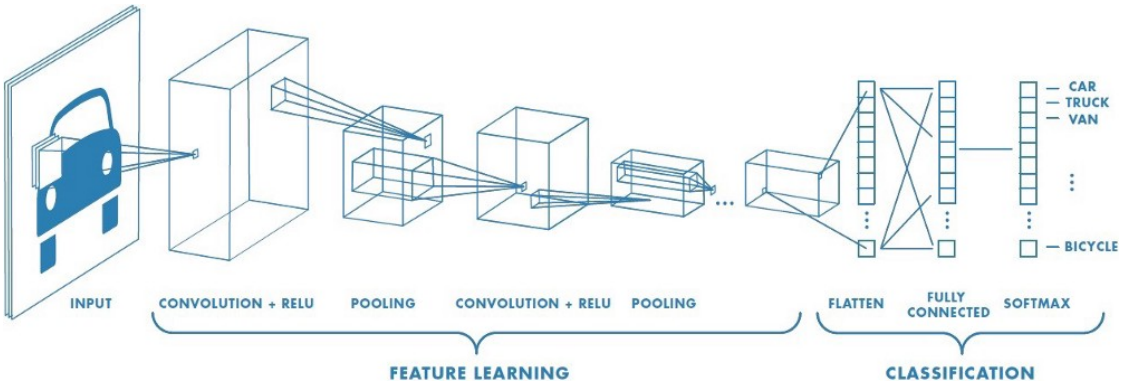


Figure 1.2: CNN architecture

The aim of this project is to give a view on application of CNNs in image classification problems. Compare results among some models using different backbones (Scratch, VGG16, ResNet34, AlexNet, SqueezeNet, MobileNet V3).

II. Methods and Architecture

2.1 Theoretical background

Each image is a matrix of pixel values. However, because images have pixel dependencies throughout (E.g: RGB, HSV, CMYK,…), we can not flatten images and feed them into a Multi-Level Perceptron for classification purposes.

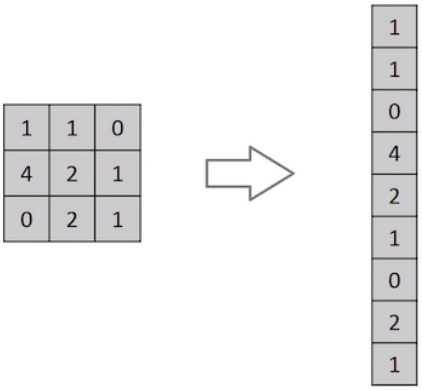


Figure 2.1: Flattening of a 3x3 image matrix into a 9x1 vector

To solve this problem, CNN is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.

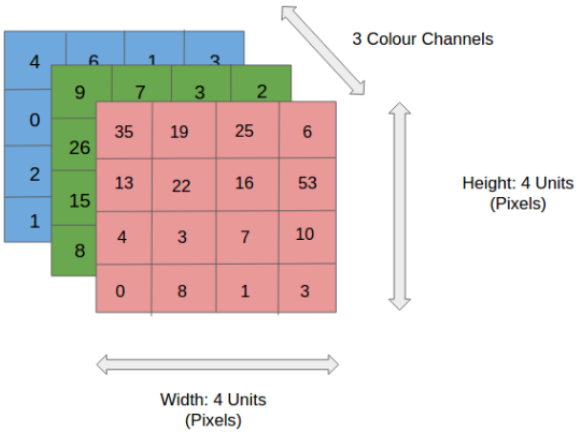


Figure 2.2: 4x4x3 RGB Image

Otherwise, once the images reach high dimensions, for example: 8K image (7680×4320), computation complexity may be extremely high. Hence, the role of the CNN is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. [3]

- **Convolution layer - the kernel**

The convolution layer is the first layer of the CNN network. Unlike a fully connected neuron, each convolutional neuron (filter) is only locally connected to the input data. The convolutional neuron slides from left to right and from top to bottom of the input data block and computes to generate an activation map. The depth of the convolutional neuron is equal to the depth of the input data block.

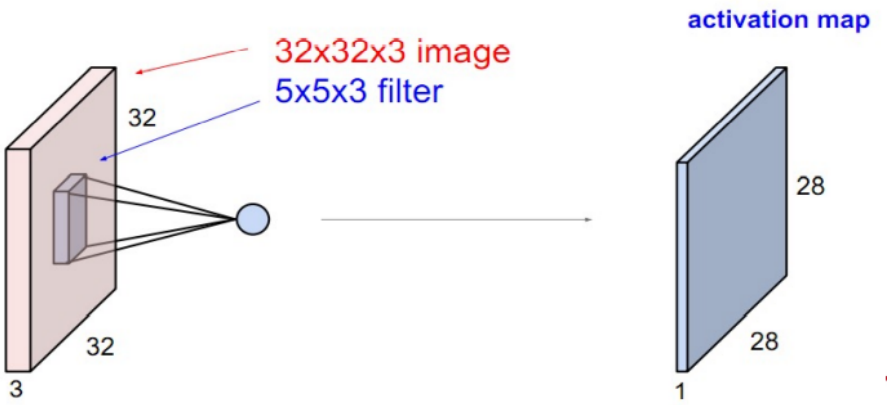


Figure 2.3: Convolution layer

A convolutional neural network is a sequence of convolutional layers stacked on top of each other, and they are interspersed with activation functions (e.g. ReLU).

- **Pooling layer**

Similar to convolutional layers, pooling is a down-sampling technique. Pooling layers help reduce the block data resolution to reduce memory and computation consumptions. Pooling layers work independently on each activation map. The max pooling layer helps the network to make the representation become invariant to the small (local) translation, or deformation (deformation-invariant) of the input data.

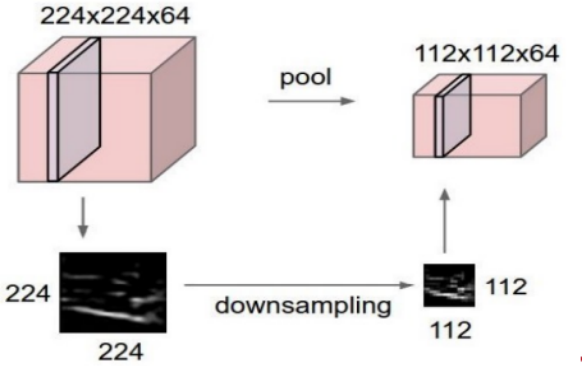


Figure 2.4: Pooling layer

- **Classification - Fully Connected Layer (FC Layer)**

After convolutional layers and pooling layers, FC Layer is a popular and cheap way to learn non-linear combinations of the high-level features which are represented by the output of the last convolutional layer. FC Layer has the same architecture as ANN. Hence, it can learn a nonlinear function in that space

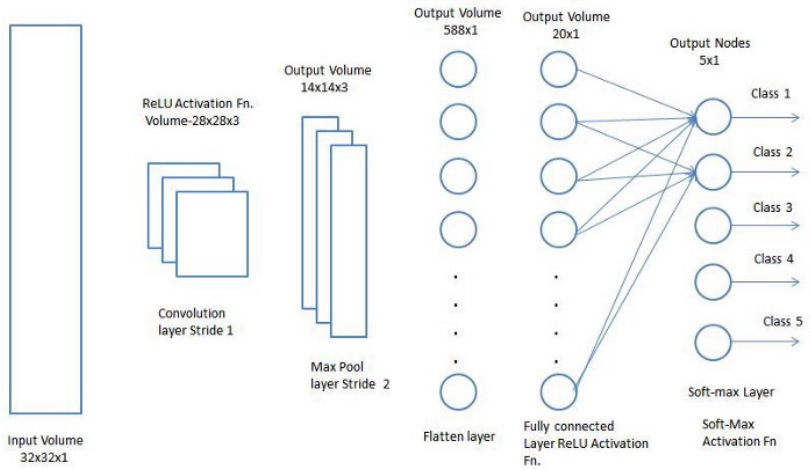


Figure 2.5: Fully Connected Layer

2.2 Architecture

As mentioned above, I built a scratch model and used 6 pretrained models for training. I also finetuned a pretrained model (SqueezeNet) and compare their results.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 16, 224, 224]             448
         MaxPool2d-2         [-1, 16, 112, 112]               0
       BatchNorm2d-3         [-1, 16, 112, 112]              32
            Conv2d-4         [-1, 32, 112, 112]           4,640
         MaxPool2d-5           [-1, 32, 56, 56]               0
       BatchNorm2d-6           [-1, 32, 56, 56]              64
            Conv2d-7           [-1, 64, 56, 56]          18,496
         MaxPool2d-8           [-1, 64, 28, 28]               0
       BatchNorm2d-9           [-1, 64, 28, 28]             128
           Conv2d-10          [-1, 128, 28, 28]          73,856
        MaxPool2d-11          [-1, 128, 14, 14]               0
      BatchNorm2d-12          [-1, 128, 14, 14]             256
           Conv2d-13          [-1, 256, 14, 14]         295,168
        MaxPool2d-14            [-1, 256, 7, 7]               0
      BatchNorm2d-15            [-1, 256, 7, 7]             512
          Dropout-16                [-1, 12544]               0
           Linear-17                  [-1, 512]       6,423,040
           Linear-18                  [-1, 256]         131,328
           Linear-19                  [-1, 128]          32,896
          Dropout-20                  [-1, 128]               0
           Linear-21                   [-1, 20]           2,580
================================================================
Total params: 6,983,444
Trainable params: 6,983,444
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 17.90
Params size (MB): 26.64
Estimated Total Size (MB): 45.12
----------------------------------------------------------------
```

Figure 2.6: Scratch model architecture and parameters

In my proposed CNN model, I built 5 convolutional layers interspersed with Batch Normalization layers and Pooling layers. I also add Dropout between fully connected layers in order to avoid overfitting. ReLU is used as an activation function in all layers except the last layer.

Beside, I used the transfer models for training (VGG16, ResNet34, AlexNet, SqueezeNet, MobileNet V3). Furthermore, a finetuned SqueezeNet model is also recommended with few adjustments in Classification.

```
(classifier): Sequential(
  (0): Dropout(p=0.5, inplace=False)
  (1): Conv2d(512, 20, kernel_size=(1, 1), stride=(1, 1))
  (2): ReLU(inplace=True)
  (3): AdaptiveAvgPool2d(output_size=(1, 1))
```

SqueezeNet Classifier

```
(classifier): Sequential(
  (0): Dropout(p=0.5, inplace=True)
  (1): Conv2d(512, 20, kernel_size=(3, 3), stride=(1, 1))
  (2): ReLU(inplace=True)
  (3): AdaptiveAvgPool2d(output_size=(1, 1))
```

Finetuned SqueezeNet Classifier

I changed kernel size from 1x1 to 3x3 and compared the results.

III. Training model

3.1 Dataset and Data preprocessing

The original dataset is Animal-10 [4] with about 26.000 images divided into 10 classes. Besides, I collected more data to have a larger dataset called “Extended animal 10” with 60.907 RGB images divided into 20 classes.

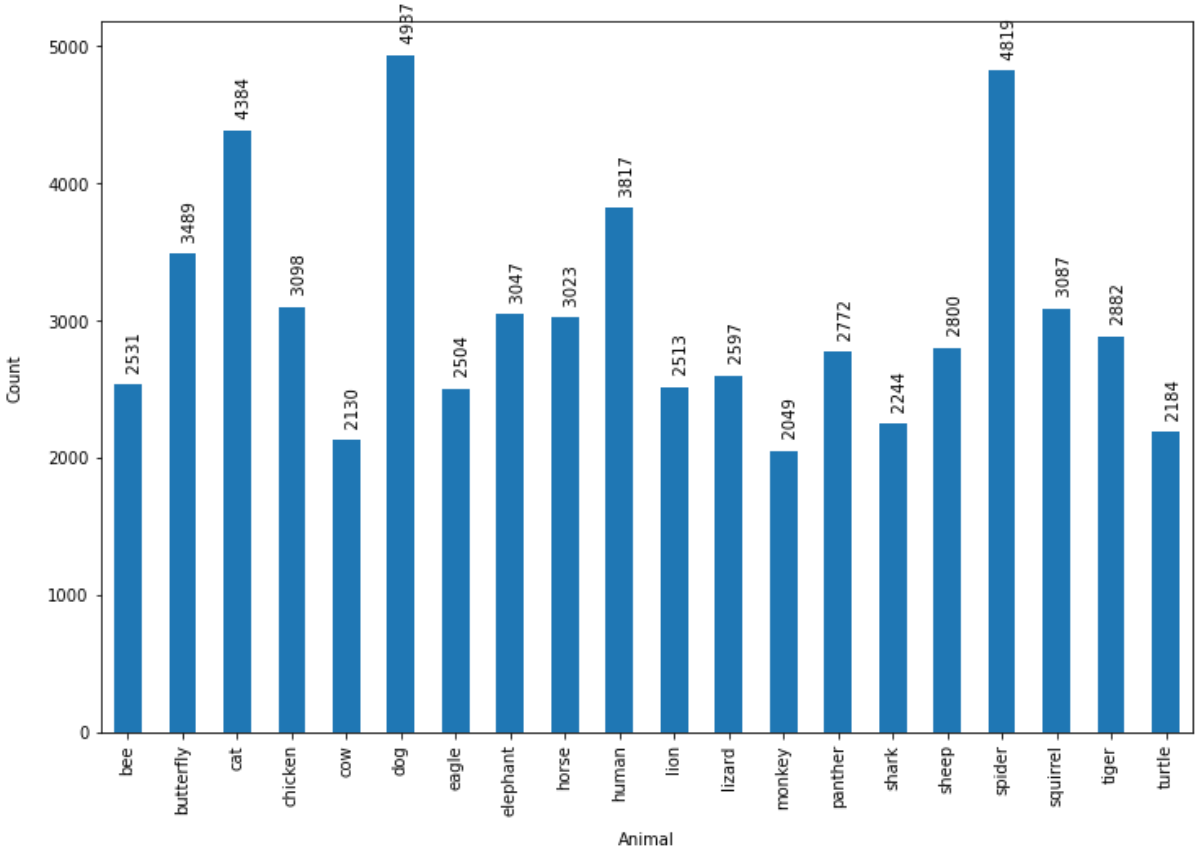


Figure 3.1: Number of images in each class

I randomly split the dataset into a training set, a validation set and a test set with proportion 70:10:20. I also resized images to 256x256, center crop images 224x224 and normalized images. Therefore, the input size of the first convolutional layers is 224x224x3.

Figure 3.2: Some resized images of each class

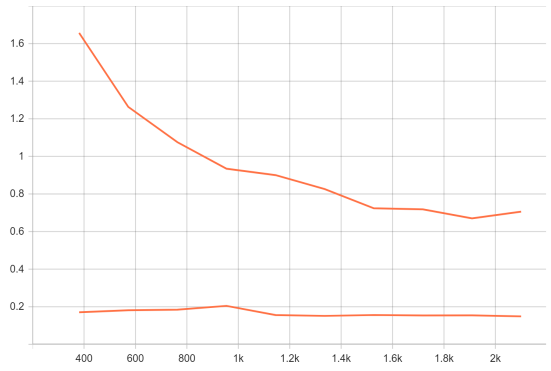
3.2 Training details

I trained the models on the training set. To measure error of models, I use Cross Entropy Loss between predicted values and actual values. To get the predicted values, I forwarded data through the network and then calculated loss at each layer by using a technique called Backward Propagation. I trained models with a learning rate equal to 0.001 and used SGD with momentum (momentum = 0.9) to update parameters. Besides, I also used a technique called Reduce Learning rate On Plateau to avoid overfitting. The learning rate will reduce if the validation loss is not improved after each 2 epochs.

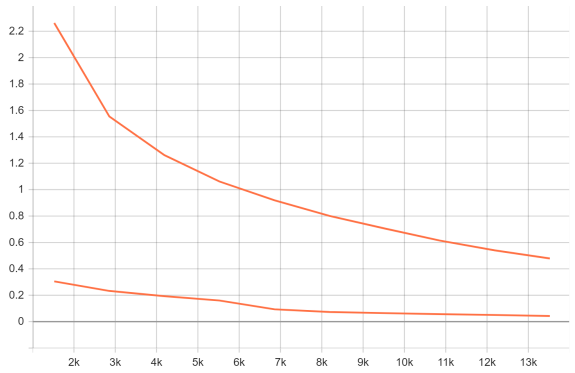
I trained all models in 30 epochs divided into 3 periods (10 epochs for each period) and captured the results of the first 10 epochs and the last 10 epochs.

- **Scratch model**

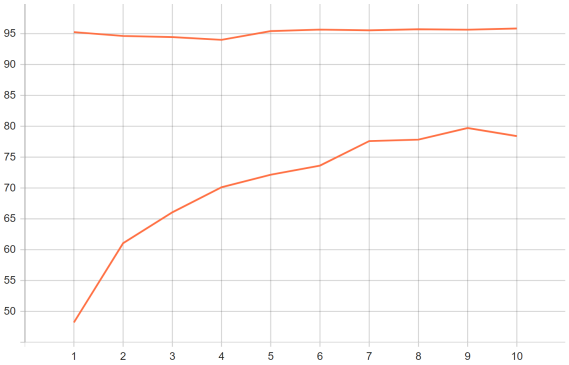
The Scratch model converges much slower than the transfer models. Hence, it took more time to have good accuracy. The figure below shows training loss, validation loss, validation accuracy of the first 10 epochs and last 10 epochs of the Scratch model.



(a)



(b)

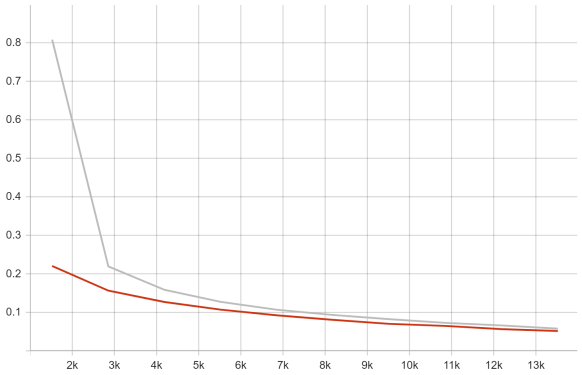


(c)

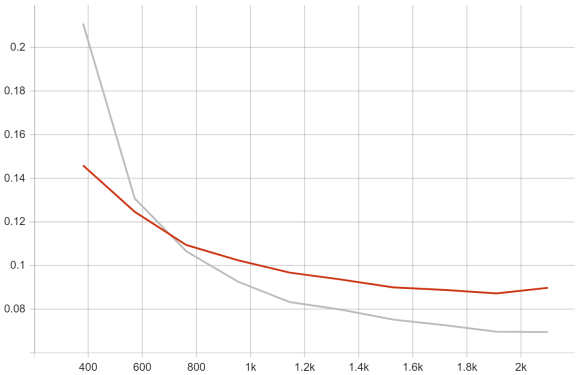
Figure 3.4: (a): Validation loss, (b): Training loss, (c): Validation accuracy

- **Pretrained models**

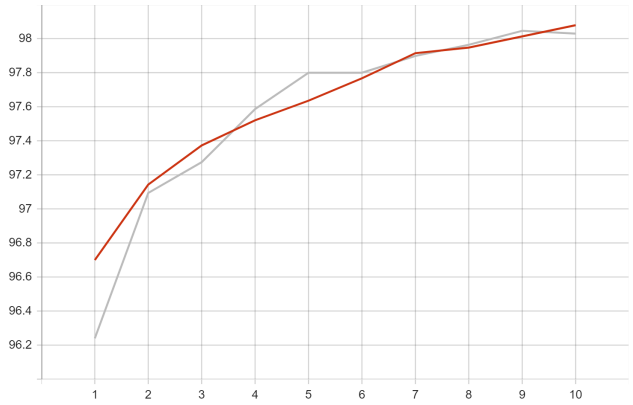
With the models having powerful backbones like VGG16 or ResNet34, the image classification problem seems to be quite easy with them. VGG16 model and ResNet34 model converge very quickly (need only 3-5 epochs to reach good accuracy) and these models tend to be overfitting if we train them for a long time. Therefore, we should stop the training process early.



(a)



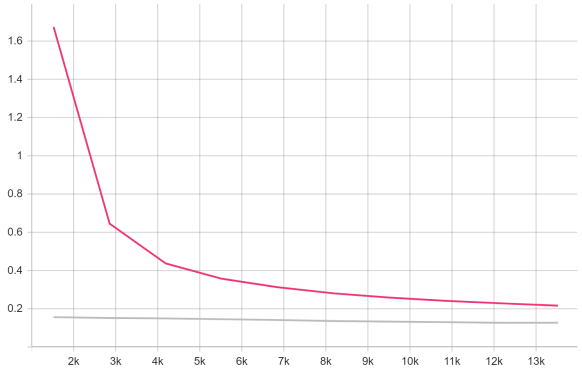
(b)



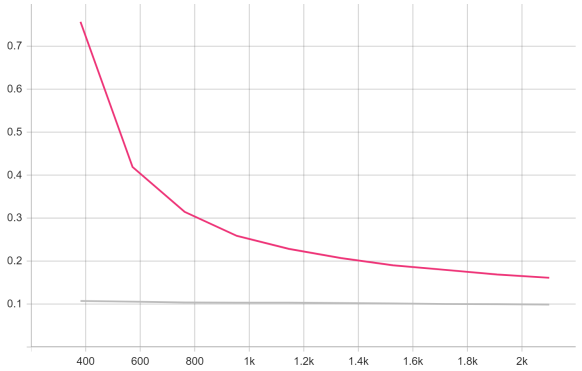
(c)

Figure 3.5: ResNet34 transfer learning (first 10 epochs and last 10 epochs)
(a): Training loss, (b): Validation loss, (c): Validation accuracy

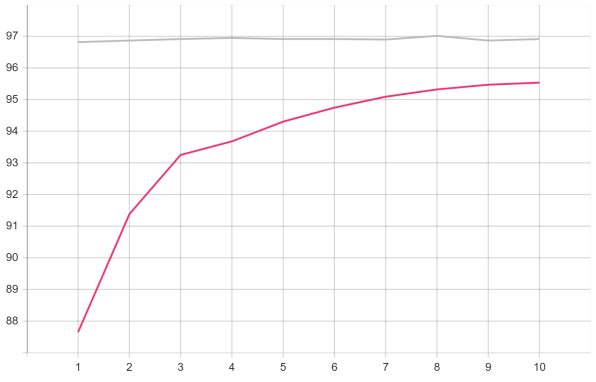
The remaining models (AlexNet, MobileNetV3, SqueezeNet) converge a little slower than the VGG16 model and ResNet34 model. However, these models still converge faster than Scratch model.



(a)



(b)

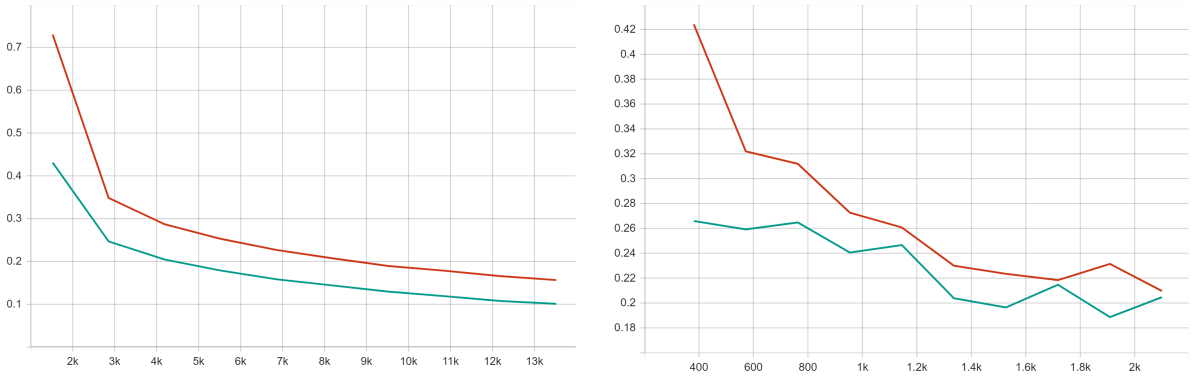


(c)

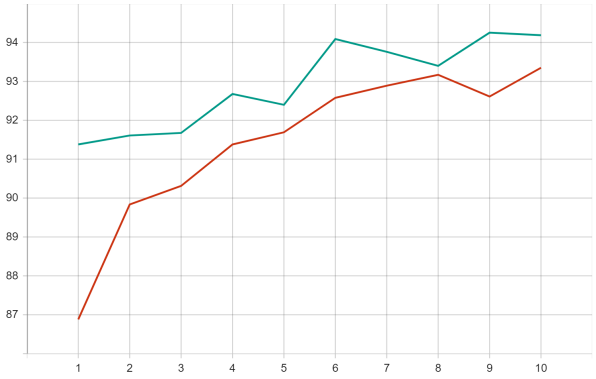
Figure 3.6: MobileNetV3 transfer learning (first 10 epochs and last 10 epochs)
(a): Training loss, (b): Validation loss, (c): Validation accuracy

- **Finetuned model**

As mentioned above, I also finetuned a SqueezeNet model. The finetuned model perform the task better than the original model.



(a) *(b)*



(c)

Figure 3.6: SqueezeNet and Finetuned SqueezeNet (first 10 epochs)
(a): Training loss, (b): Validation loss, (c): Validation accuracy

IV. Results and evaluation

- **Test accuracy, test loss and training time of the first 10 epochs**

Model	Test loss	Test accuracy	Training time
Scratch	0.665180	79% (9678/12183)	2h 20m 12s
AlexNet	0.196851	94% (11457/12183)	1h 14m 33s
MobileNet V3	0.179037	94% (11571/12183)	1h 37m 11s
SqueezeNet	0.223354	93% (11336/12183)	1h 16m 33s
Finetuned SqueezeNet	0.214708	93% (11432/12183)	2h 13m 58s
VGG16	0.109355	96% (11793/12183)	3h 30m 39s
ResNet34	0.072701	97% (11937/12183)	2h 33m 58s

- **Test accuracy, test loss and training time of the last 10 epochs**

Model	Test loss	Test accuracy	Training time
Scratch	0.143667	96% (11710/12183)	2h 12m 59s
AlexNet	0.103296	96% (11814/12183)	1h 43m 49s
MobileNet V3	0.087669	97% (11899/12183)	3h 28m 55s
SqueezeNet	0.139064	95% (11663/12183)	2h 36m 27s
Finetuned SqueezeNet	0.116006	96% (11738/12183)	2h 36m 14s
VGG16	0.102806	97% (11826/12183)	3h 31m 54s
ResNet34	0.069605	98% (11960/12183)	3h 25m 27s

The test accuracy of the models having strong backbones like VGG16 and ResNet34 is very high while the training time for 10 epochs is quite long. The other finetuned models have test accuracy lower than VGG16 and ResNet34 (accuracy is still good enough) but their training time is quite low. The result of Scratch model after 10 epochs is not really good. However, after 30 epochs, the Scratch model's test result is improved.

- **Some test results**

Figure 4.1: Some test results

- **Confusion matrix**

After testing the models, I have a confusion matrix to represent the number of true values in each class of the test set.

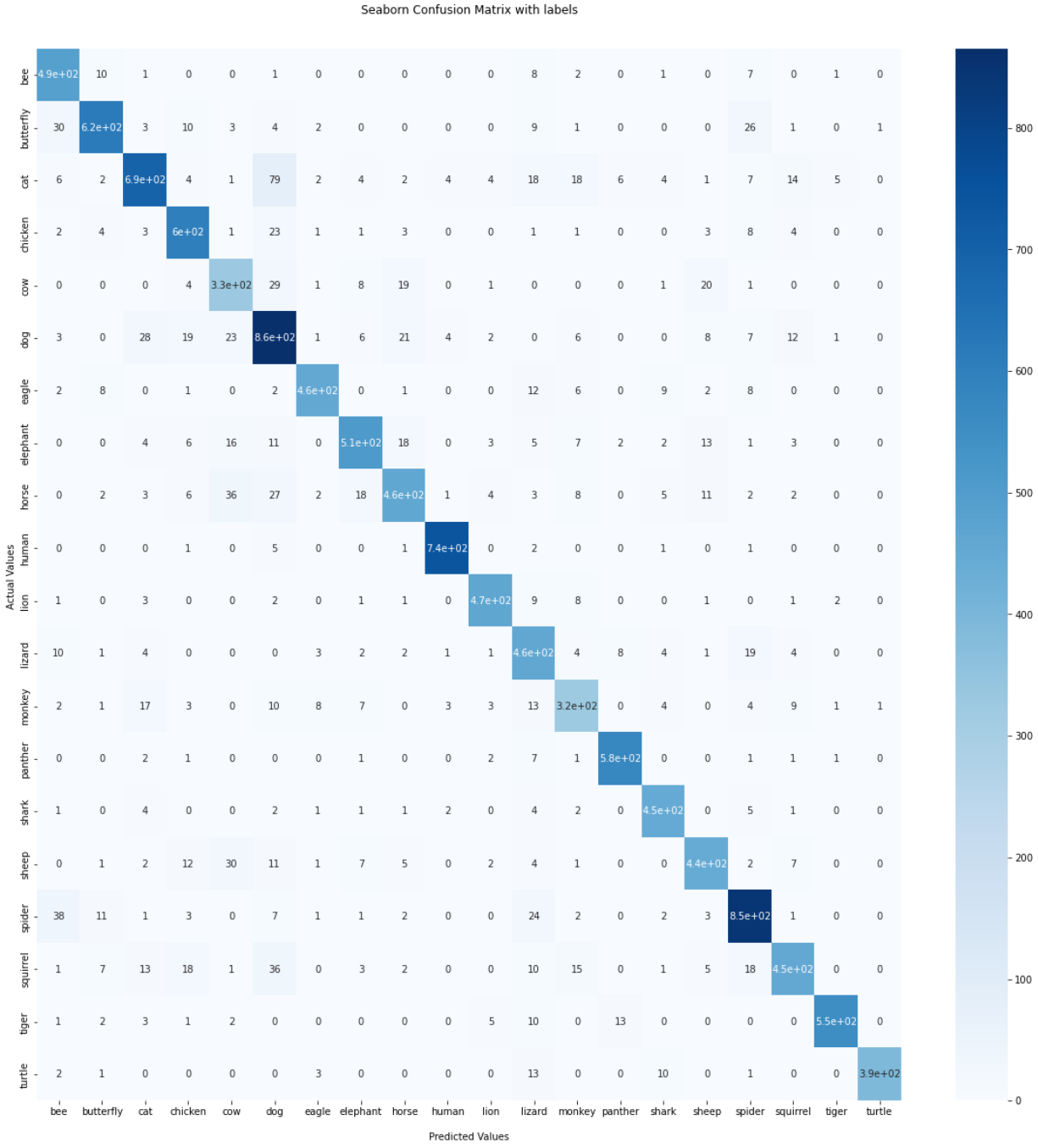


Figure 4.2: Confusion matrix of Scratch model after 10 epochs

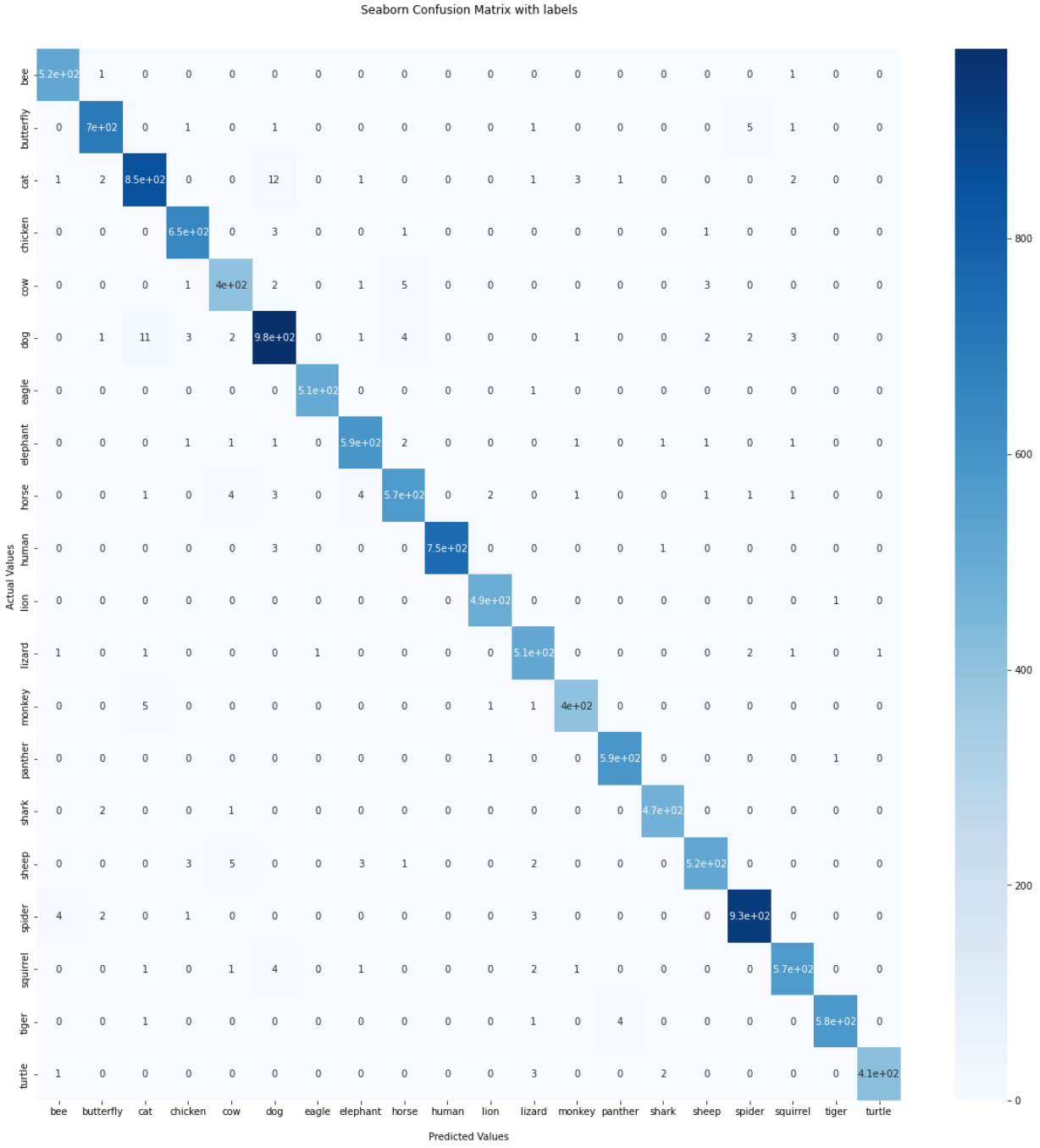


Figure 4.3: Confusion matrix of Scratch model after 30 epochs

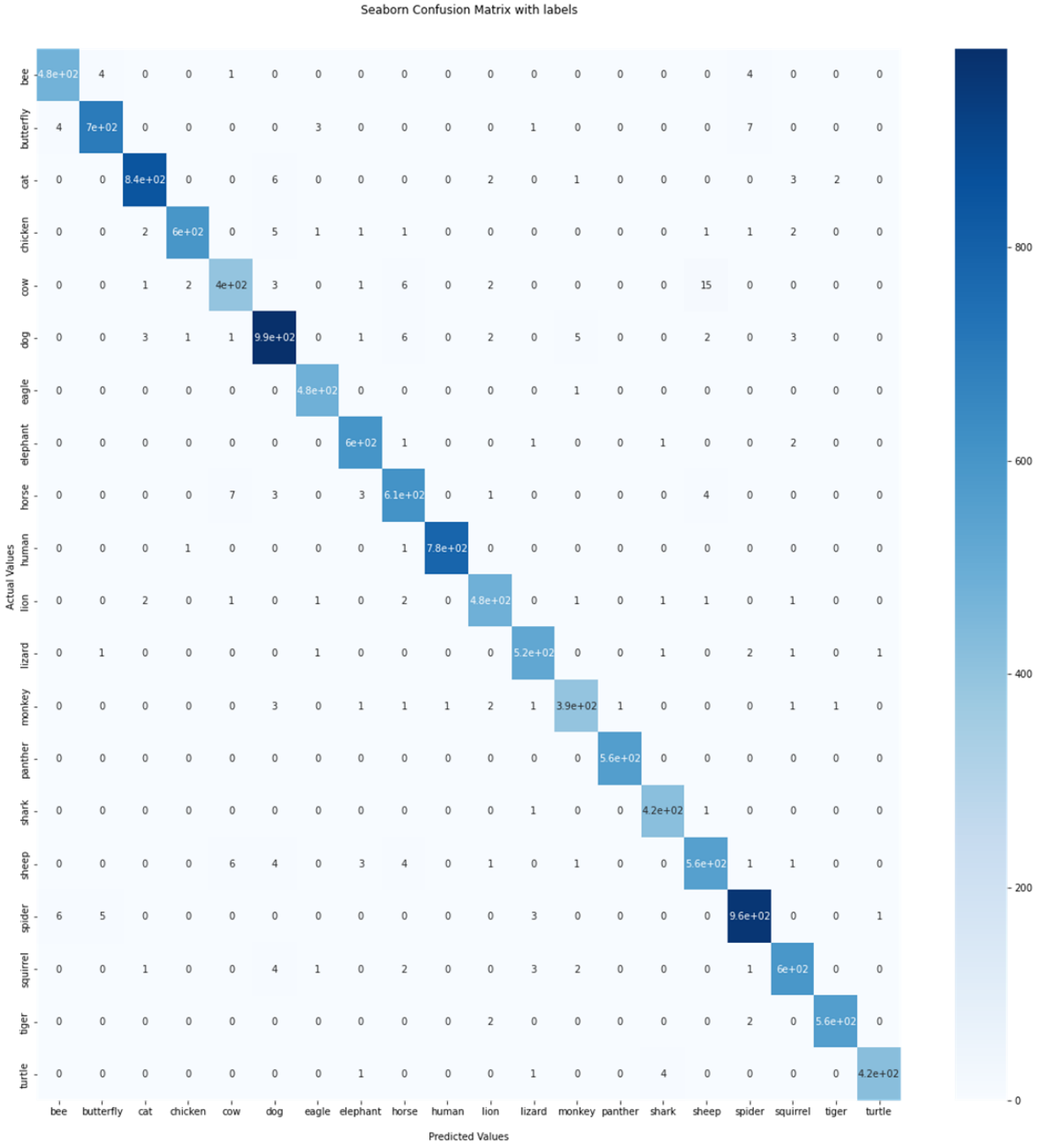


Figure 4.3: Confusion matrix of VGG16 model after 30 epochs

V. Technology, framework and hardware used

Programming language: Python 3.9

Frameworks and libraries: Pytorch, torchvision, tensorboard, matplotlib, seaborn, pandas, numpy

Environment: Anaconda, Jupyter Notebook

Hardware: CPU: Intel(R) Core(TM) i7 - 1075H @ CPU 2.60GHz
GPU: NVIDIA GeForce GTX 1650 Ti

VI. Conclusion and future improvement

In this project, I applied the theory of Convolutional Neural Network to build a model from scratch and use some pretrained models for training. To evaluate the models, Cross Entropy Loss is used. I trained models with a learning rate equal to 0.001 and used SGD with momentum (momentum = 0.9) to update parameters. Besides, I also used a technique called Reduce Learning rate On Plateau to avoid overfitting.

Generally, all models perform quite well after 30 epochs of training. However, the transfer models tend to be overfitting after 30 epochs. Hence, we should stop training early in practical application. With the Scratch model, because of converging slower than transfer models, we need to train it longer.

Through this project, I have learned about the application of CNN to solve image classification problems. Otherwise, the dataset used in this project seems to be quite easy for the models. Therefore, I need to train my models in the "harder" dataset (outlier, blur images, slanted images, …) and use some image/data augmentation techniques to check the effectiveness of models.

References

[1] https://cs231n.github.io/convolutional-networks/

[2] https://wiki.tino.org/convolutional-neural-network-la-gi/

[3] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[4] https://www.sciencedirect.com/science/article/pii/S0924271617303660?casa_token=5JEQa5dcfhIAAAAA:TX3Z5EGekN22wyMtP8ZMljoW4JGQ5CWNB_ekk58ssb5y3jmviEmpWGGYNKlJGblivh3v_whEPw

[5] (PDF) Image Classification Using Convolutional Neural Networks (researchgate.net)

[6] Kaiming He & Xiangyu Zhang & Shaoqing Ren & Jian Sun. "Deep Residual Learning for Image Recognition". In: (2015).